东南大学电工电子实验中心

实 验 报 告

课程名称：数字电路实验

第 5 次实验

实验名称：FPGA 时序逻辑设计

院 （系）：电气工程学院专 业：电气工程及其自动化

姓 名：王皓冬 学 号：16022627

实 验 室：401 实验组别：

同组人员： 实验时间：12 月 26 日

评定成绩： 审阅教师：

一、实验目的

1. 综合前面所学的各项内容
2. 了解掌握数字系统设计的流程和方法
3. 掌握复杂电路连接和调试技能

二、实验原理

0 输入信号与输出信号编码

输入信号

用 COL[3..0]作为输入，COL[i]代表模拟键盘第 i 个列信号。

输出信号

用 pout[3..0]、dout[6..0]和 ROW[3..0]作为输出。其中，pout[i]标识第 i 个七段数码管；dout[j]标识七段数码管的第 j 个灯管，低电平时灯亮（由 seg-decoder 代码推导出，在此仅提及）；ROW[k]标识模拟键盘的第 k 个行信号。COL 与 ROW 共同组合为键盘功能。

其他标识说明

1. $(A_3\ A_2\ A_1\ A_0)_2$、$(B_3\ B_2\ B_1\ B_0)_2$ 分别代表 A、B 对应的四位二进制数，A 代表被操作数，B 代表操作数，如减法计算表达式为 A - B。“1”代表该为二进制数为 1。number 代表数据，输出“1”代表有数据 0~9 输入。
2. ADD、SUB、MUL 分别是 addition、subtraction、multiplication 的缩写，分别代表加法、减法、乘法。“1”代表选中该计算符，“0”代表未选中。operator 代表计算符，输出“1”代表+、-、x 号被输入。
3. equal 代表等号，输出“1”时代表等号符被输入。
4. 角标 i 代表输入，o 代表输出，分别是 in、out 的缩写。
5. ones-dig 代表 BCD 码个位，tens-dig 代表 BCD 码十位，dig 是 digit 的缩写。
6. Y[i]代表总状态机的第 i 个状态，具体状态说明见主控电路 State 的状态转换图。“1”时代表处于该状态，“0”代表未处于。
7. CLK 代表时钟信号，c0 代表经转换为 10kHz 的时钟信号。
8. clean 代表清零，输出“1”代表执行清零功能，处于“0”时不作用。CLRN 代表经取反后的 clean 信号，是因为某些芯片清零引脚以低电平有效，为了统一 clean 信号高电平有效而引入的。

实验内容：

1. 实验基础部分(70%) 设计一个简易计算器，它具有下列运算功能：

(1) 一位十进制数的相加；

(2) 一位十进制数的相减；

(3) 数值和运算符用 4×4 键盘输入（实验室提供接口程序），其中 A 为“+”，B 为“-”，C 为“×”，E 为“=”

(4) 数值用数码管以十进制形式显示，以加法为例，初始时显示全“0”，先输入被加数，再输入运算符，按下运算符键后，数码管显示全“0”，再输入加数，方法和前面一样，最后按下“=”，数码管显示运算结果

2. 实验扩展部分(30%)

(1) 一位十进制数的相乘，必须采用串行乘法实现；

(2) 其他自选功能

3. 实验要求：

(1) 根据设计要求划分设计层次、单元模块和接口信号，在预习报告上记录设计过程，绘制系统框图，每个模块的状态转移图或 ASM 图，并设计验证方案。

(2) 用原理图输入法设计所有单元模块并编译，分析编译时产生的错误和警告信息

(3) 对所有的单元模块进行功能仿真，并记录和分析全部仿真结果

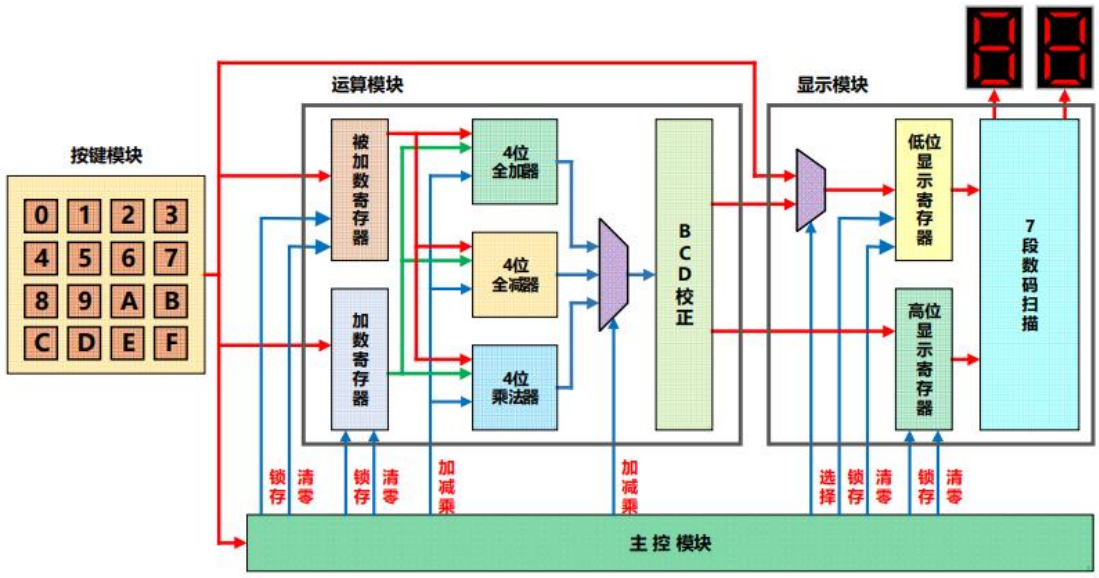
(4) 在顶层文件中连接全部单元模块并编译、综合、分配管脚和适配。

(5) 对整个系统进行时序仿真，并记录和分析仿真结果。

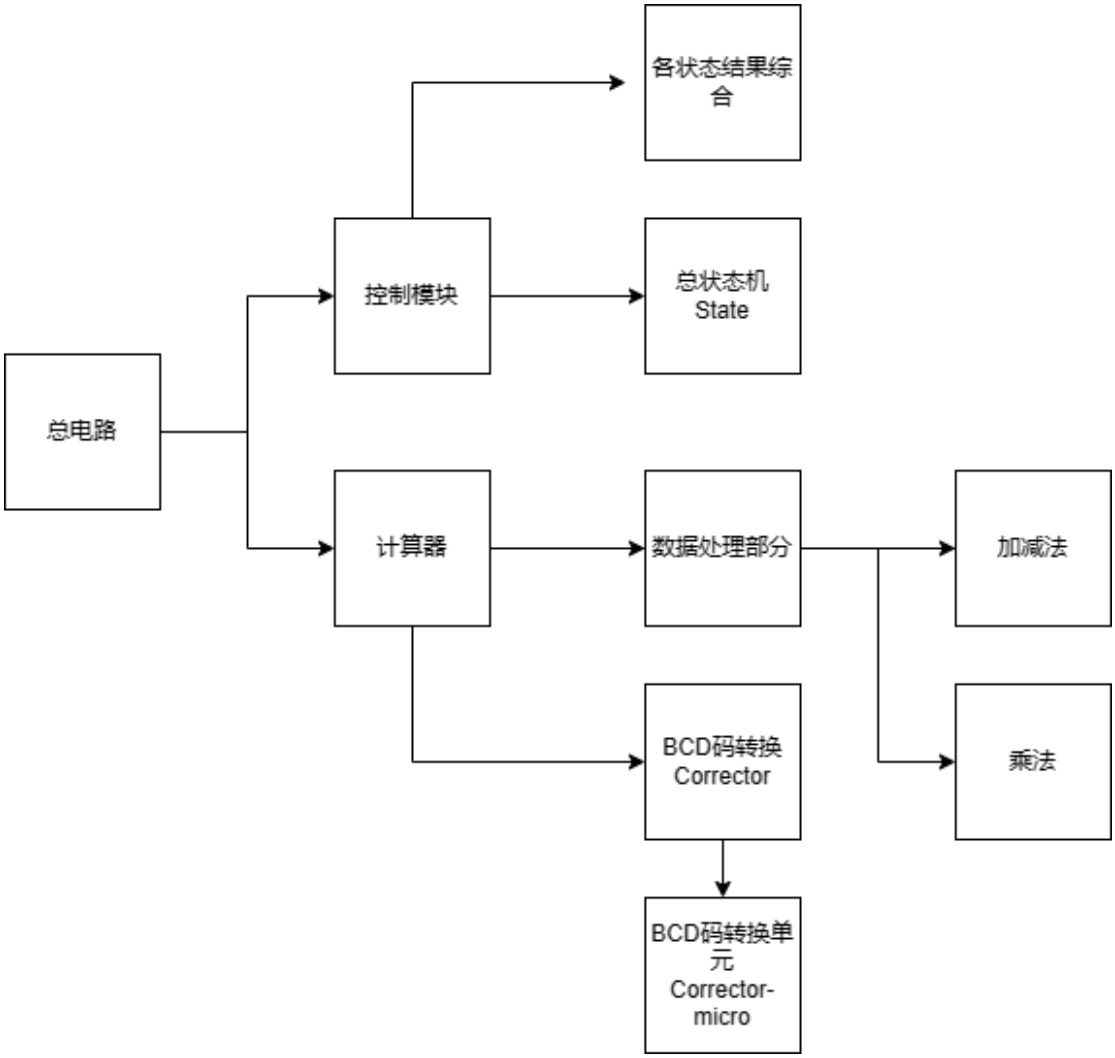
(6) 将仿真正确的设计下载到实验箱上，连接输入输出设备和示波器进行板级验证

1. 逻辑模块

该计算器的逻辑模块可按如下树状图表示:



粗略分为控制电路与数据处理电路两模块。控制电路包括了整个流程的状态机控制功能，数据处理模块包括了计算器的功能系统与校正系统。具体模块细分如下图。



在进行本实验时，采取了由下至上的设计思路，从模块单元逐一验证，最终组装为顶层电路。

2. 模块设计与状态图

加减法功能：

加减功能直接采用了实验二中设计的 BCD 加减器的设计，其真值表也引用了该实验实验报告的对应部分真值表。如下:

表 1 加法功能

输入										输出					
A3	**A2**	**A1**	**A0**	**B3**	**B2**	**B1**	**B0**	**CI**	**ASi**	**ASo**	**S4**	**S3**	**S2**	**S1**	**S0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0			0	0	0	0	1
0	0	0	0	0	0	0	1	0			0	0	0	0	1
0	0	0	0	0	0	0	0	1			0	0	0	0	1
0	0	0	1	0	0	0	1	0			0	0	0	1	0
0	0	0	1	0	0	0	0	1			0	0	0	1	0
0	0	1	1	0	0	0	1	0			0	0	1	0	0
…	…	…	…	…	…	…	…	…			…	…	…	…	…

0	1	0	1	0	1	0	0	1			1	0	0	0	0
0	1	1	0	0	1	0	1	0			1	0	0	0	1
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

表 2 减法功能

输入										输出					
A3	**A2**	**A1**	**A0**	**B3**	**B2**	**B1**	**B0**	**CI**	**ASi**	**ASo**	**S4**	**S3**	**S2**	**S1**	**S0**
0	0	0	0	0	0	0	0	0			0	0	0	0	0
0	0	0	1	0	0	0	0	0			0	0	0	0	1
0	0	0	0	0	0	0	1	0			1	0	0	0	1
0	0	0	0	0	0	0	0	1			1	0	0	0	1
0	0	0	1	0	0	0	1	0			0	0	0	0	0
0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	1	1	0	0	0	1	0			0	0	0	1	0
…	…	…	…	…	…	…	…	…			…	…	…	…	…
0	0	0	1	1	0	1	0	1			1	1	0	1	0
0	0	1	0	1	1	0	1	0			1	1	0	1	1
…	…	…	…	…	…	…	…	…		…	…	…	…	…	…

该模块无状态机。

乘法功能（Mul4）：

是通过串联乘法实现的。设计思路同教材。

在设计之初，采用的设计是 4 个 7 位二进制加法器串行实现，其中 7 位二进制加法器可以通过 4 位全加器得到。完成设计后反思时，我对比了自己的思路与教材思路，得到：

1. 该思路***难以实现清零功能***。虽然可以 input 一个 clean 信号，并接过一个反相器后与所有结果作与运算，使得 clean 输入为高电平时能将输出强行置零，但其清零时长难以控制。
2. ***无寄存功能***。这是最主要的一个区别，反思我自己的设计思路时，发现如果输入信号临时改变，会导致输出对应改变。

乘法功能，或计算器功能的***清零条件***将在下文***寄存功能***处具体分析。

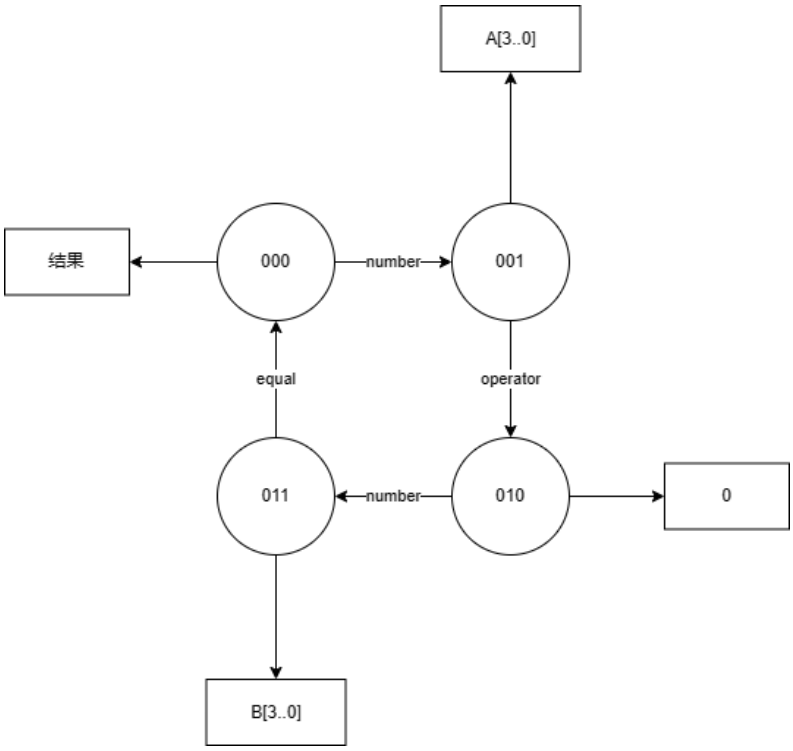
主控电路（State）：

设计时，对于“达到条件才进入下一个状态”有多种思路：

1. 可以采用 74161，在使能端附加门电路组实现。由于过于复杂，未采用。由此引出一个话题：所有有转移条件的计数器，我们都可以采用在计数器***使能端附加电路***的方式实现功能。下文的思路也是基于此的。
2. 可以采用上一个实验中的 One-Hot 码或二进制码序列检测器的设计。同样，该设计较为复杂，故也未采用。
3. 可以利用数选器 74151 实现不同状态判断不同条件。记 74151 的各输入数据下标为 i，将D_i接入状态 i 对应跳转条件（由 State 给出），如D_0接入 number。将计数器的二进制计数反接到 74151 的地址端，实现选择条件的功能；再将 151 的正输出 Y 接到 161 的使能端，控制是否计数到下一个状态（地址），便可实现 151 与 161 的***互锁控制***。

这种方法较为简单，实际设计的时候采用的是这种方法。

对应状态转换图如图所示。



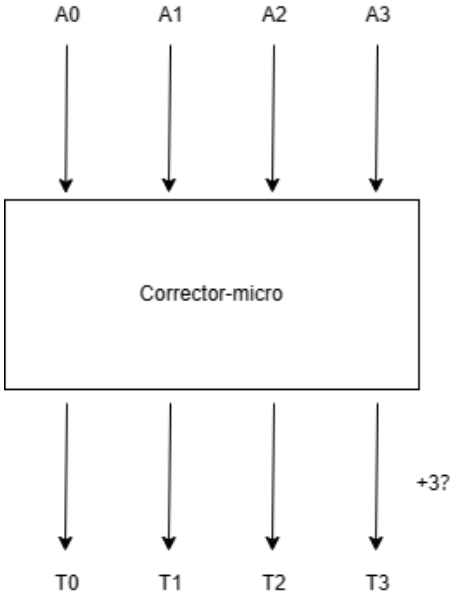
其中方框中内容为数码管应当显示的内容。

8 位二进制码转 BCD 功能（Corrector-micro，Corrector）：

是利用校正单元（Corrector-micro）错位串联实现的，而校正单元采用的原理是“输入的 4 位二进制码大于等于 5 则加 3”。这种方法的本质原理是，5（0101）左移一位，相当于乘 2，得到的 BCD 码应是 10（1000），但实际是 1010（二进制码的 10）。因此对 0101 加 3 得到 1000，转换为 BCD 码。这其实是余三码的类似思路。

4 个转换单元错位串联，即可实现移位校正。

逻辑图如下。

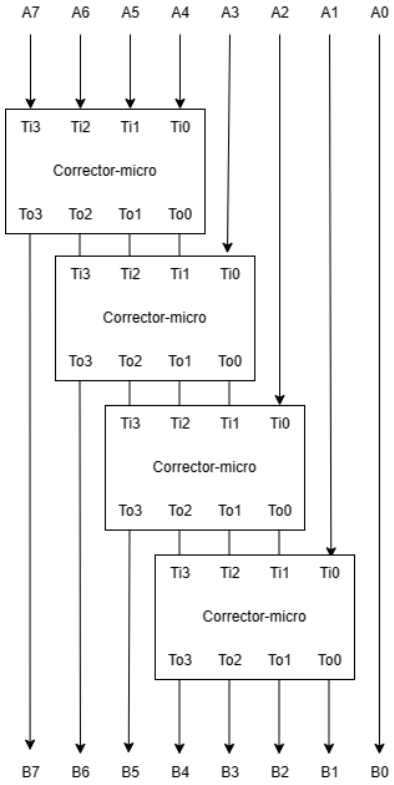


移位是在将 Corrector-micro 级联时完成的。该电路不需要状态转换机，可直接得出结

果。这是因为乘法模块***已有寄存功能***，该模块只需要完成将乘法的二进制输出转换为 BCD 码输出的功能。

而判断“大于等于 5”可以通过译码的思路实现，译出的十进制数端口中$\overline{Y}_5$~$\overline{Y}_9$的端口输出有效电平即代表输入的二进制数大于等于 5。级联两个 74138，得到 4 线-16 线译码器。将输入的 4 位二进制数接入译码器端口，得到译码结果。将低位 138 的结果$\overline{Y}_5$~$\overline{Y}_7$，以及高位 138 的结果$\overline{Y}_0$~$\overline{Y}_1$作与非运算，即可判断“大于等于 5”。这里没有考虑 10~15 的数据，因为这些输入属于***无效输入***，按键只会出现 0~9 的数据，不会出现这种情况。

将校正器单元错位组合为总校正器逻辑图如下。



数码管显示：

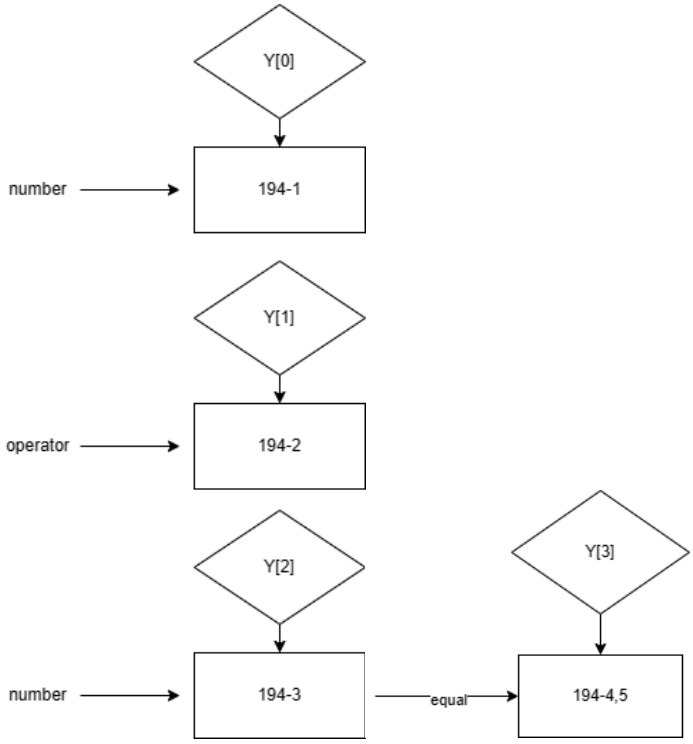
由于计算器输出仅为计算结果，而数码管应实时显示输入数 A、B 和结果。因此，需要设计一个模块用于在不同 State 状态下正确输出对应的数值。

该模块的思路很简单，只需要将对应 State 状态的函数 Y[i]与数值作与运算，再将所有状态的结果作或运算即可。

同样，由于已有寄存功能，该模块无状态机。

寄存功能：

利用 74194 寄存。采用 5 个 74194，分别存放 A、符号、B、BCD 输出。其状态转换图如下。



其中，菱形框图代表执行判断。

这里的清零功能需要特别判断。由于该模块只进行数据存储，不参与运算，我们采用假设法分析：假设没有条件清零功能，我们先后输入两个计算式：

$$A_1\ operator_1\ B_1 = C_1$$

$$A_2\ operator_2\ B_2 = C_2$$

那么第二个计算式刚输入时，计算器模块会首先计算第一个算式的数据B_1,与第二个计算式的已输入数据A_2，得到的结果是

$$A_2\ operator_1\ B_1 = C$$

这时计算器结果是错误的，输入到符号时同理。而当且仅当状态 3，即已输入$A_2\ operator_2\ B_2$时，这时计算器的输出是正确的C_2，而寄存器 194-4，5 中存放的是C_1。因此，只需要在状态 3 时将乘法功能的数据清零，使其重新运算即可（此时的运算结果恰是正确的）。因此，寄存部分***不需要清零功能***。

该分析也补全了外电路中***计算器模块的清零条件***。

减法“负号”的处理：

由于实验二中设计的 BCD 加减器符号位的负号输出是 1，这样就会得到“0 – 1 = 11”的奇怪结果。因此，负号需要特殊处理。

最开始采用了全置 1 的处理方式。由于减法功能结果范围为[-9，9]，且 BCD 码单位的范围为[0，9]，因此将 BCD 码的十位全置 1，即 1111 便不会输出数值。本实验的 seg_decoder 模块是根据 16 进制设计的，1111 对应的输出是 F。因此，这种设计思路下，输出的负号用 F 代替，“0 - 1”输出结果为“F1”。

然而，这种结果仍有悖常识，人机交互性不强，仍需要特殊说明。因此，试探性对其优化，使得负数结果能直接输出负号。

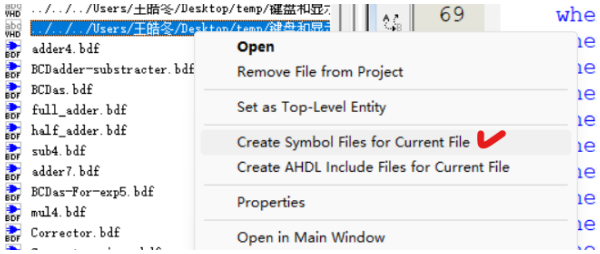
实验中提供的 seg_decoder 是.vhd 文件，是代码。翻译代码，代码中 7 段译码管“1111”

对应的代码是 011100（F），“0000”对应的代码是“000001”，则说明“0”代表数码管灭。因此“-”对应的代码应是“111110”（0 和-的数码管灯恰好是相反的）。作更改如下。

```
81      when "1100" => reg_dout <= "0110001"; -- "C"  0x31
82      when "1101" => reg_dout <= "1000010"; -- "D"  0x42
83      when "1110" => reg_dout <= "0110000"; -- "E"  0x30
84      when "1111" => reg_dout <= "1111110"; -- "F"  0x38//original state:0111000
85      when others => reg_dout <= "0110000"; -- "E"  0x30
86    end case;
```

其中，origin state：0111000 是为了便于改回原代码添加的注释。

通过.vhd 文件创造芯片：



即可实现正确输出负号。

3. 逻辑函数式

由于电路主要为中规模逻辑电路组合，因此在这里仅列出部分引脚的逻辑函数式。

总电路状态机（State）：

计数器两使能端记为 ENT 与 ENP，151 输出为 Y，则

$$ENT = ENP = Y$$

151 的地址端应有

$$D_0 = D_2 = D_4 = number$$

$$D_1 = operator$$

$$D_3 = equal$$

寄存模块：

用于储存第 i 个状态信息的 194 两使能端记为$S_0[i]$、$S_1[i]$，则应有

$$S_0[i] = S_1[i] = Y[i]$$

串行乘法器（Mul4）：

储存低位数据的 194 数据输入端记为 LOW[3..0]，其使能端记为S_{0L}、S_{1L}，输出记为 P[3..0]，右移端记为 SRlow；高位对应记为 HIGH[3..0]与S_{0H}、S_{1H}，输出记为 T[3..0]。

加法器输出记为 S[3..0]与 Co，两加数分别为 Ai[3..0]、Bi[3..0]。

乘法结果输出为 M[7..0]。

则 194 数据输入端应有

$$LOW[i] = B[i]$$

$$HIGH[j] = \begin{cases} S[j+1] & j < 3 \\ Co & j = 3 \end{cases}$$

使能端应有

$$S_{0L} = Y[5]$$

$$S_{1L} = Y[0]$$

$$S_{0H} = S_{1H} = Y[5]$$

这里 Y[i]代表计数器-译码器组合的第 i 个译码输出，Y[5]的下标 5 的原因将在电路原理图部分结合电路图样阐释。

整体输出应有

$$M[3..0] = P[3..0]$$

$$M[7..4] = T[3..0]$$

加法器输入应有

$$Ai[3..0] = T[3..0]$$

$$Bi[3..0] = A[3..0] \cdot P[0]$$

上两式中等号右侧逻辑表达式可对换位置。

最后，两 194 间的移位：

$$SRlow = S[0]$$

校正器单元（Corrector-micro）：

记 4 线-16 线译码器的第 i 个译码输出为 YiN，判断数据大于等于 5 的信号为 judge，则

$$judge = \sum_{i=5}^{9} \overline{YiN}$$

加法器的+3 输入（0011）记为 Three[3..0]，则该输入端逻辑变量为

$$Three[i] \cdot judge$$

BCD 校正器（Corrector）：

已在设计思路部分给出了具体组合逻辑。

BCD 计算器（Calculator）：

合并三种计算结果的思路同 BCD 加减器，即将功能输入与上数值再作并运算。记 BCD 加减器输出为 SAB[7..0]，BCD 乘法器输出为 SM[7..0]，则个位输出有

$$ones\_dig[3..0] = ADD \cdot SAB[3..0] + SUB \cdot SAB[3..0] + MUL \cdot SM[3..0]$$

十位由于对负号进行了特殊处理，记全高的四位二进制数为 high[3..0]，有

$$tens\_dig[3..0] = ADD \cdot SAB[7..4] + SUB \cdot high[3..0] + MUL \cdot SM[7..4]$$

显示模块：

记状态 Y[i]对应显示的内容应为 Disp[i][3..0]，总显示为 Disp[3..0]（个位为例，十位同理），则

$$Disp[3..0] = \sum Disp[i] \cdot Y[i]$$

其中 i=2 是输入计算符跳转得到的状态，应显示 0，因此 i=2 时的 Disp[2][3..0]是用四根接地线组成的 Zero[3..0]（0000）。

顶层（cal-UL）：

按各模块功能设计接线即可。其中，如设计思路中所述，乘法器（也就是封装完成的计算器）

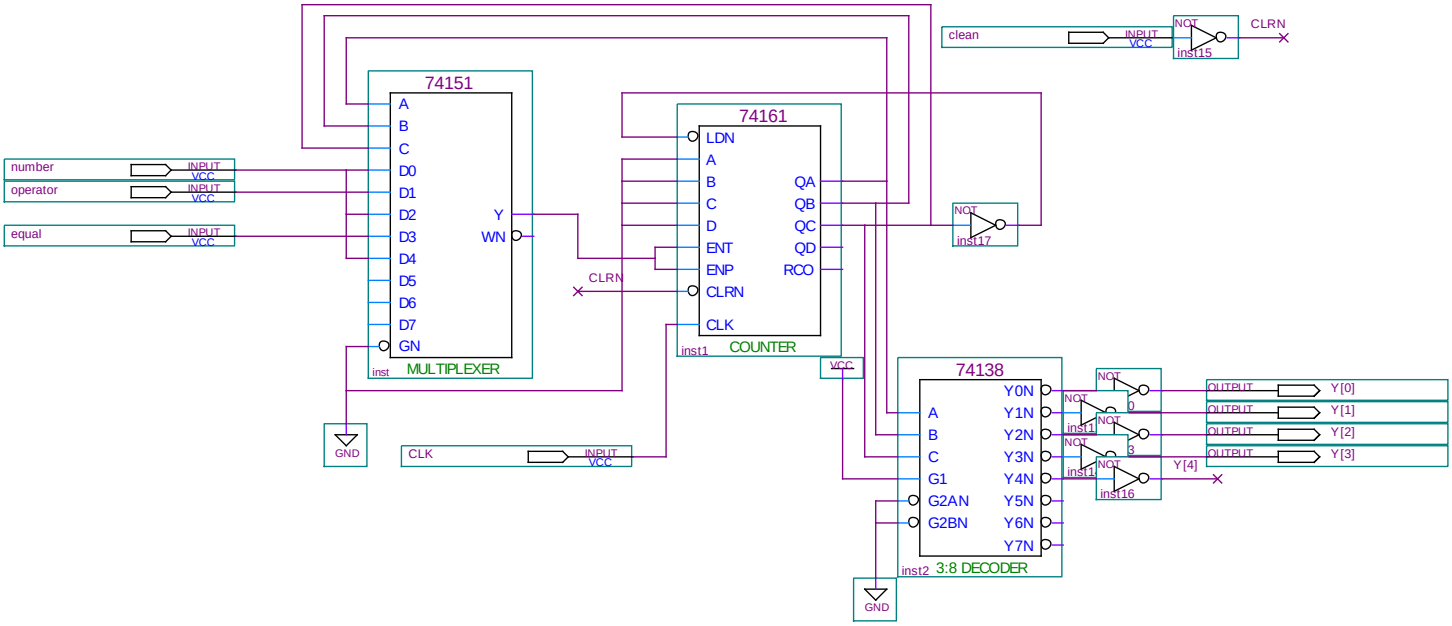
的清零端输入记为 clean-2，应使其在“完成一次计算后再次输入计算式的加数时”清零。因此得到

$$clean\_2 = Y[2] \cdot number$$

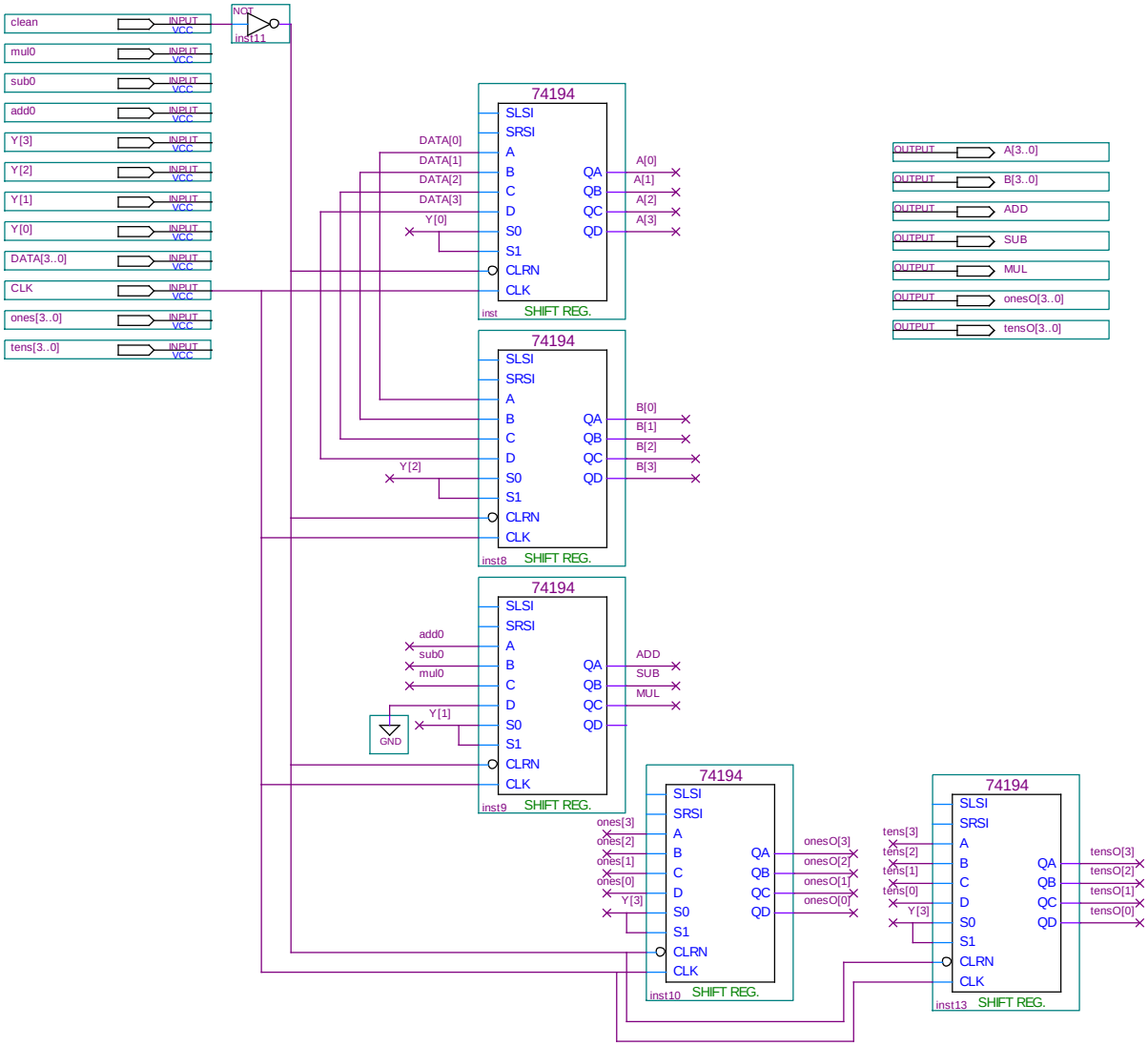
4. 电路原理图

BCD 加减器的电路在实验二中已经说明，且较为简易，这里不再赘述。

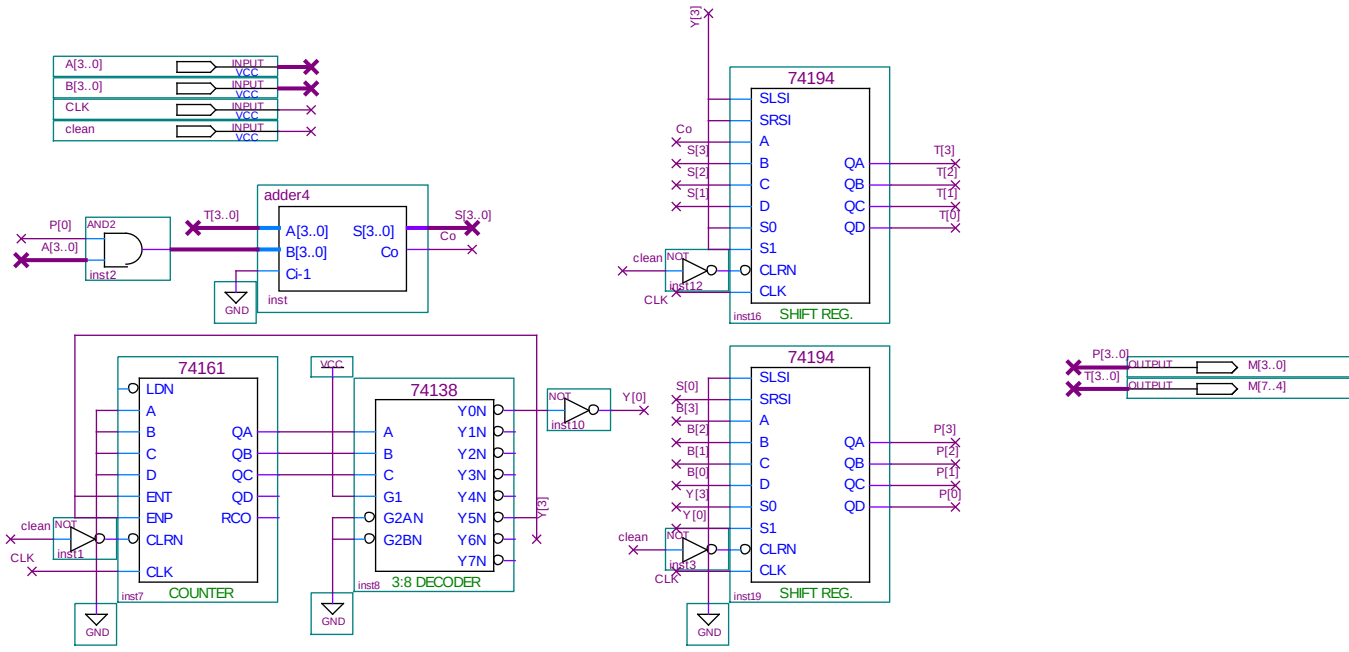
总电路状态机（State）：



寄存模块：

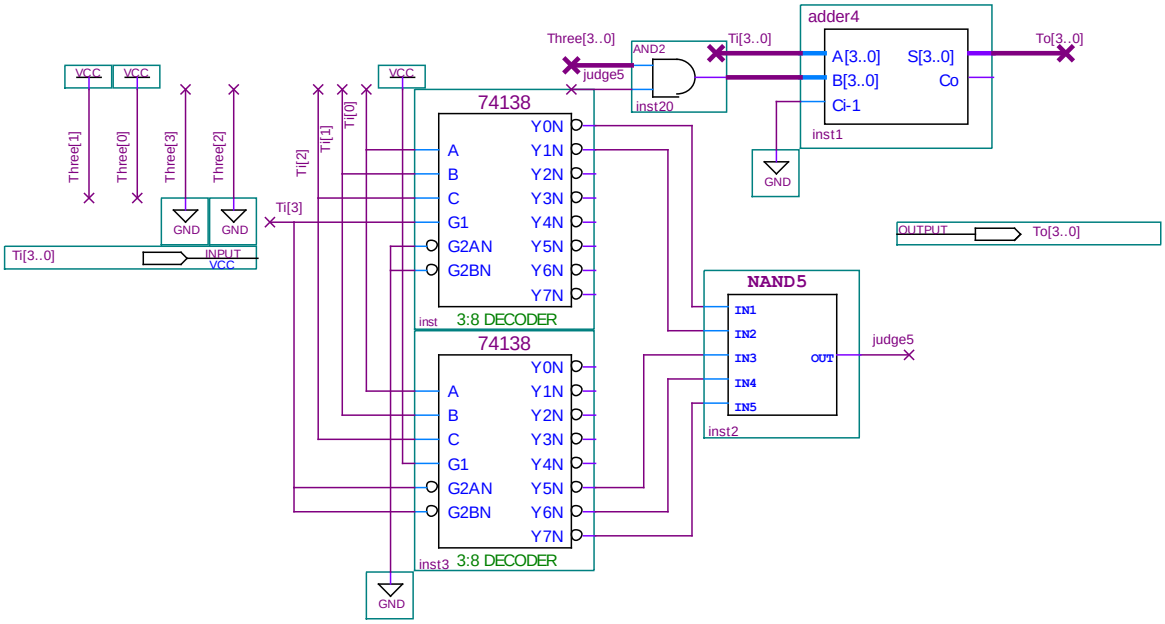

clean
mul0
sub0
add0
Y[3]
Y[2]
Y[1]
Y[0]
DATA[3..0]
CLK
ones[3..0]
tens[3..0]
INPUT
VCC
NOT
inst111
74194
SLSI
SRSI
A
B
C
D
S0
S1
CLRN
CLK
QA
QB
QC
QD
DATA[0]
DATA[1]
DATA[2]
DATA[3]
A[0]
A[1]
A[2]
A[3]
SHIFT REG.
inst
B[0]
B[1]
B[2]
B[3]
inst8
ADD
SUB
MUL
GND
inst9
onesO[3]
onesO[2]
onesO[1]
onesO[0]
inst10
tensO[3]
tensO[2]
tensO[1]
tensO[0]
inst13
OUTPUT
A[3..0]
B[3..0]
ADD
SUB
MUL
onesO[3..0]
tensO[3..0]

串行乘法器（Mul4）：

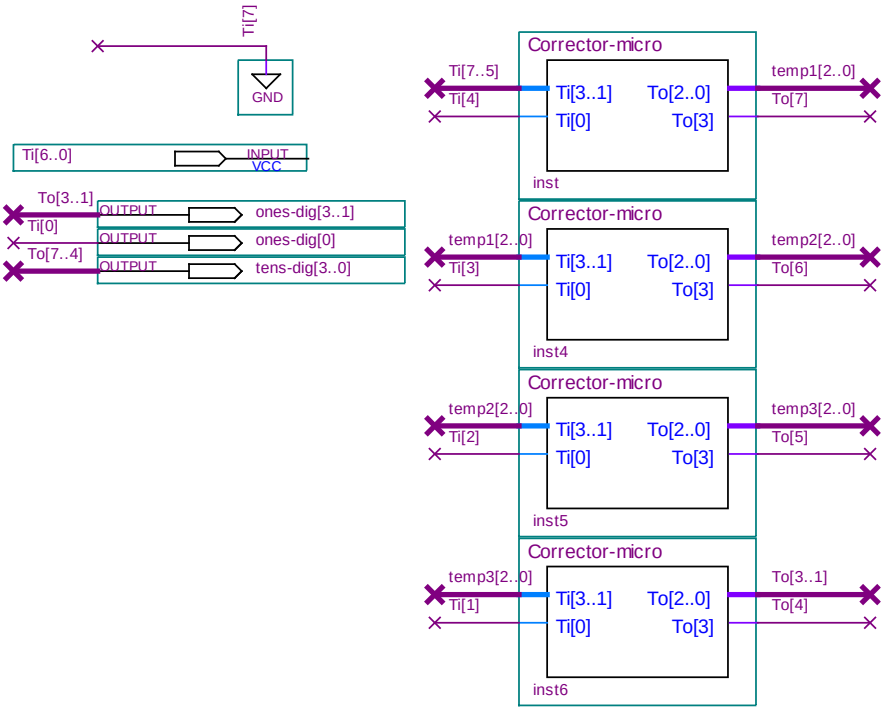

A[3..0]
B[3..0]
CLK
clean
INPUT
VCC
P[0]
AND2
inst2
adder4
A[3..0]
B[3..0]
Ci-1
S[3..0]
Co
T[3..0]
inst
GND
Y[3]
74194
SLSI
SRSI
A
B
C
D
S0
S1
CLRN
CLK
QA
QB
QC
QD
S[3]
S[2]
S[1]
T[3]
T[2]
T[1]
T[0]
NOT
inst12
inst16
SHIFT REG.
74161
LDN
ENT
ENP
RCO
COUNTER
inst7
inst1
74138
Y0N
Y1N
Y2N
Y3N
Y4N
Y5N
Y6N
Y7N
G1
G2AN
G2BN
3:8 DECODER
inst8
inst10
Y[0]
S[0]
B[3]
B[2]
B[1]
B[0]
P[3]
P[2]
P[1]
P[0]
inst3
inst19
P[3..0]
T[3..0]
M[3..0]
M[7..4]

其中，由于 CLK 控制计数器间接控制 74194 使能端，与 CLK 直接接入 74194 之间存在延迟关系；且加法器的输入有赖于高位 74194 的输出数据变化。因而实际计数归零、开始计数变为 1 的瞬间，74194 仍然未变化。所以上图设计时，存在两个 CLK 的延迟，计数器的结束位为 5。

校正器单元（Corrector-micro）：

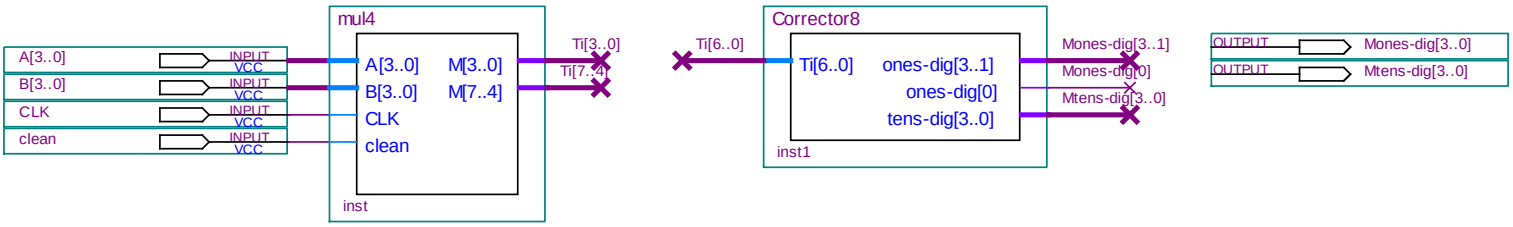


8 位 BCD 校正器（Corrector8）：



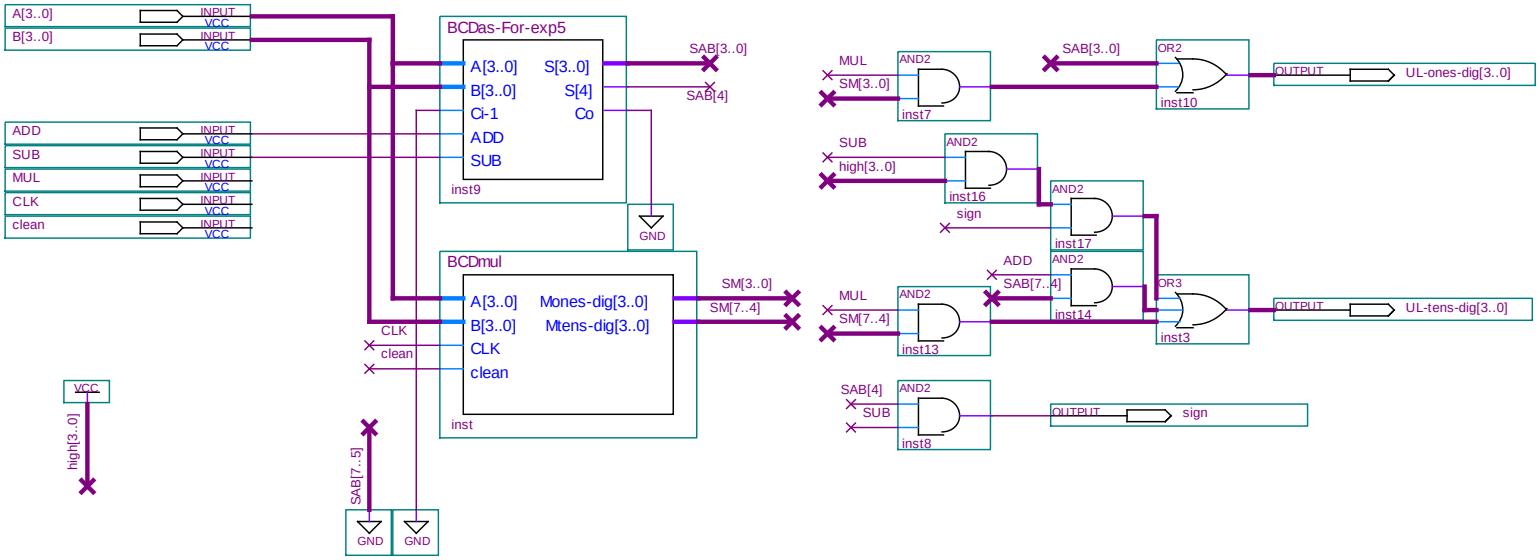
该校正器是在 Corrector-micro 的基础上完成的，即校正器单元的封装芯片。

BCD 乘法器（BCDmul）：



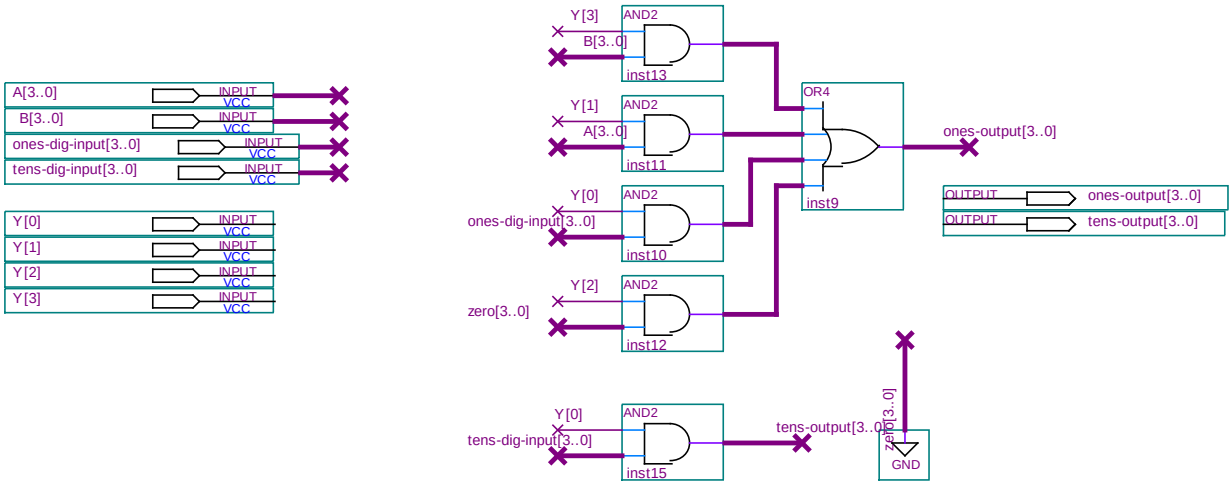
两模块简单顺联即可。

BCD 计算器（calculator）：

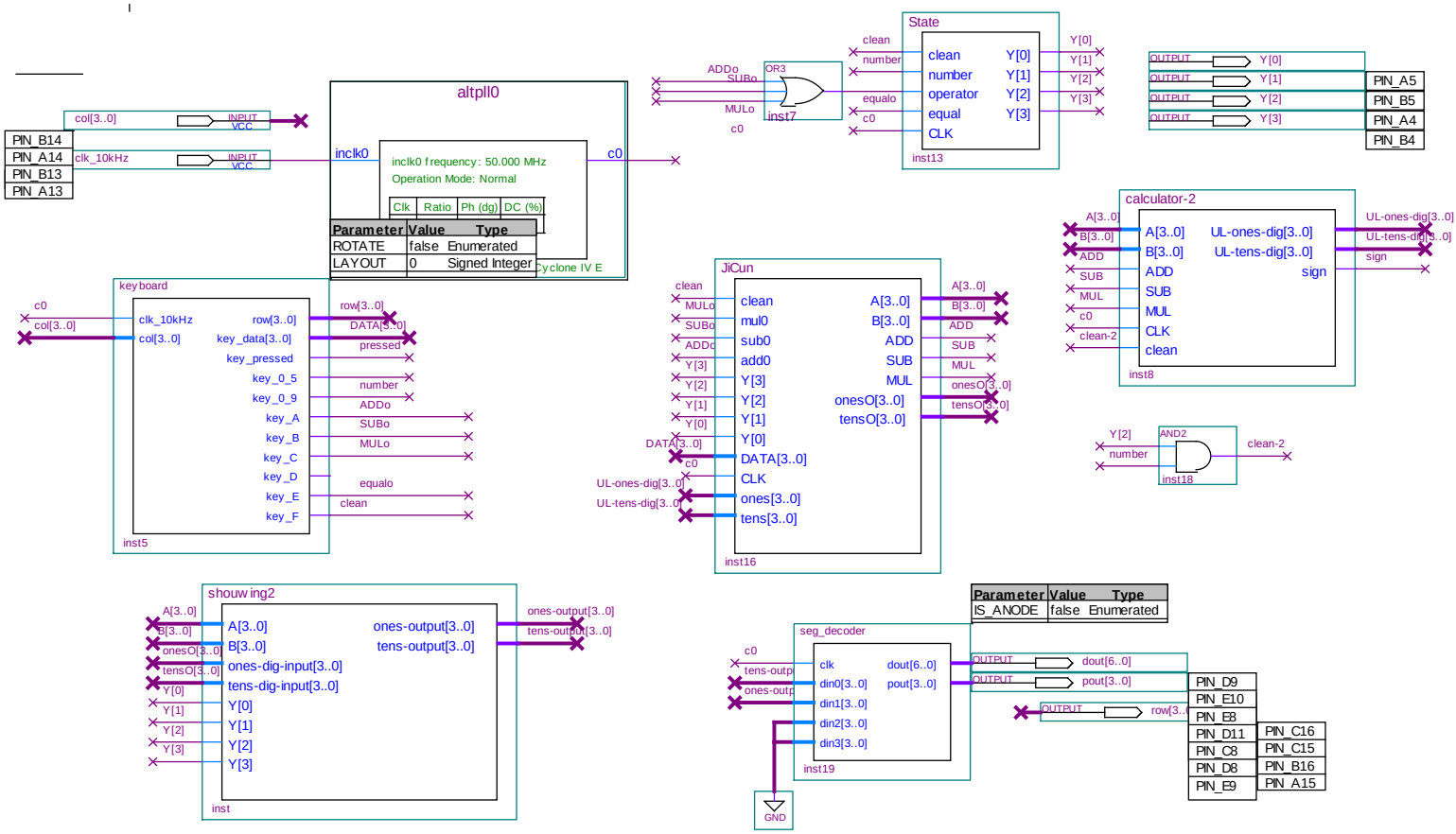


如上。其中，为了便于扩展功能，对 8 位输出的高四位与低四位及符号位分别输出。

显示模块：



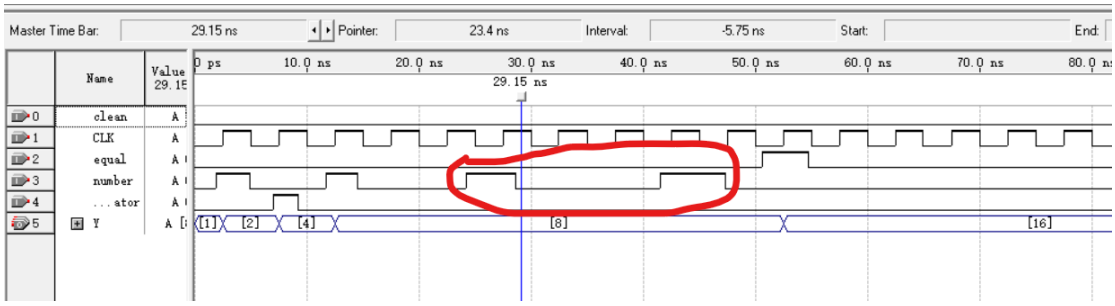
顶层（cal-UL）：



5. 仿真

部分模块，如寄存器模块过于简易，验证通过即可，不需要仿真，因此没有记录仿真。

总电路状态机(State)：

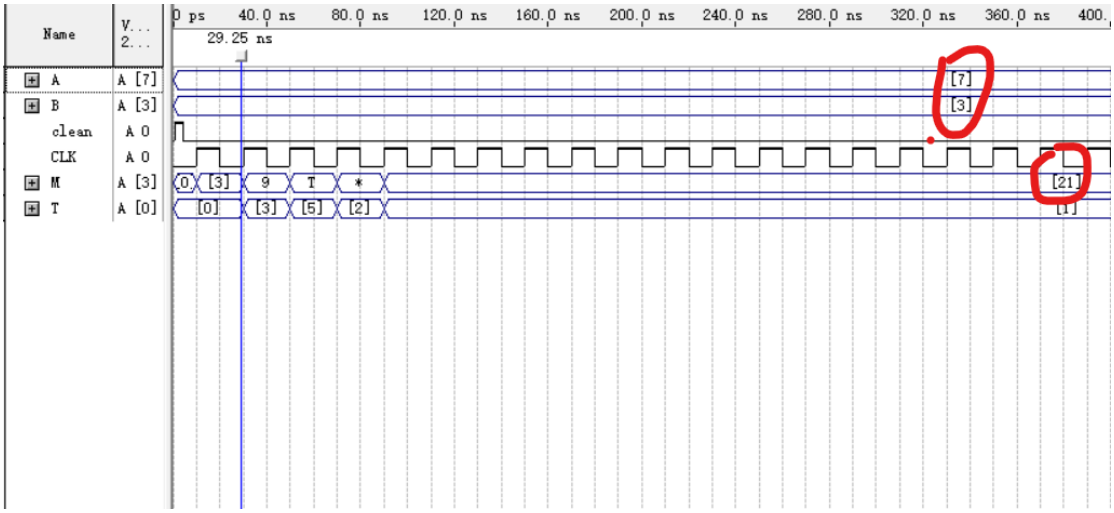


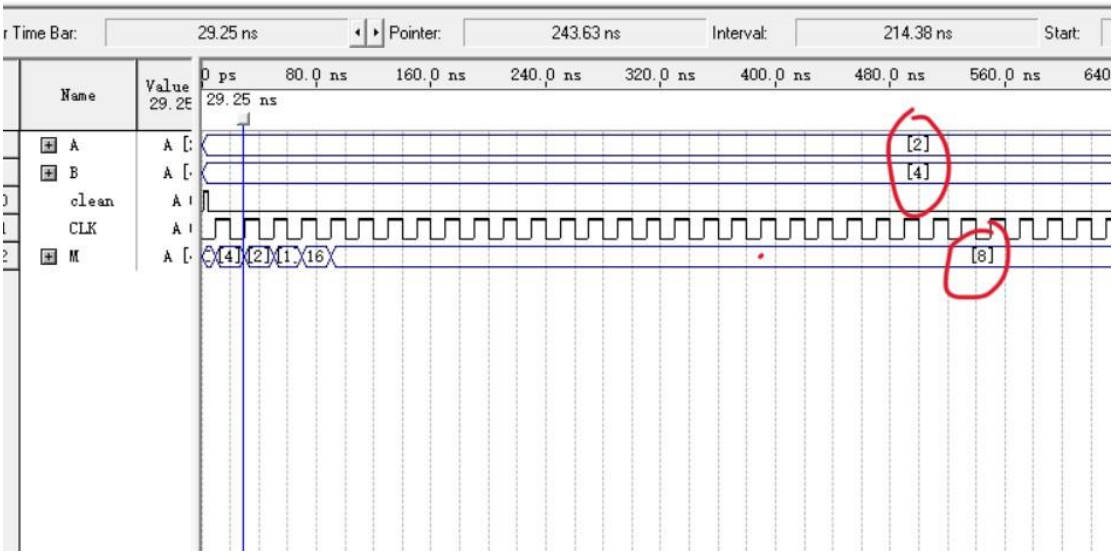
可以看到，状态机正确进行了状态转换。其中，Y 初始状态为 1 的原因是 Y 是利用 One-Hot 码表征电路状态的，初始状态是 Y[0]输出为 1，Y[5..0]=00001，故初状态输出 1。

红色部分为验证错误输入的部分。可以看到，应当输入等号（equal）时我们进行错误输入（这里为 number），状态机不会进入下一个状态。

串行乘法器(Mul4)：

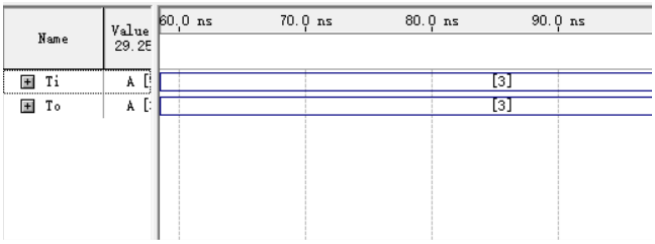
仿真所用计算式为 7 x 3 = 21 与 2 x 4 = 8。M 为输出，T 为高位寄存器的输出端数值，是便于验证引入的。

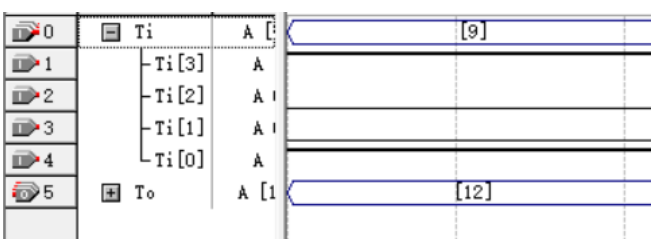


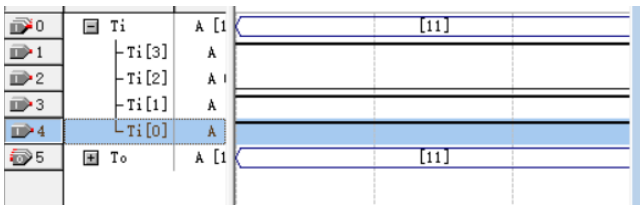


可以看到，在几次移位后，乘法器正确输出了结果的二进制表示，并能够保持。为方便验证功能，在 t=0 时为 clean 添加了一个高电平脉冲。

校正器单元(Corrector-micro)：



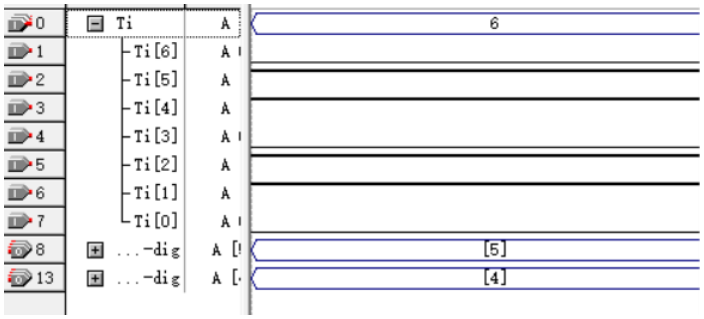




可以看到，数据为 3 时不校正；数据为 9 时校正为 12；数据为 11 时超出校正范围，不校正。

8 位 BCD 校正器(Corrector8)：

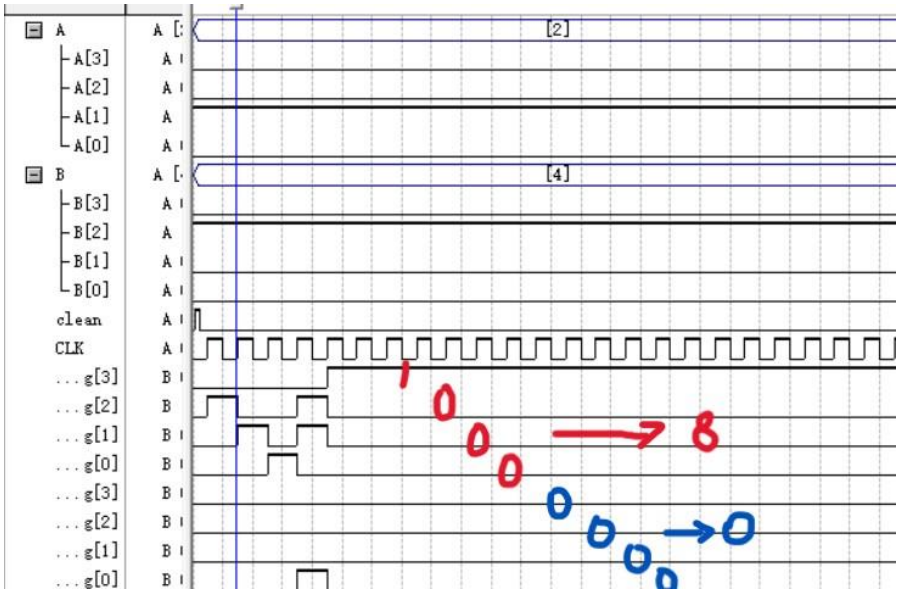
以 54（0011 0110）为例。图中显示的 Ti 为“6”是错误的，因此截取了完整的波形表征是二进制数据 54。



可以看到，十位和各位分别输出了 5 和 4，结果正确。

BCD 乘法器(BCDmul)：

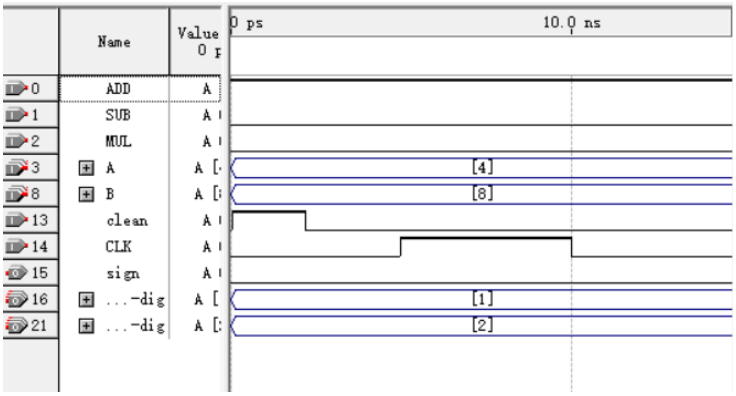
以 2 x 4 = 8 为例。



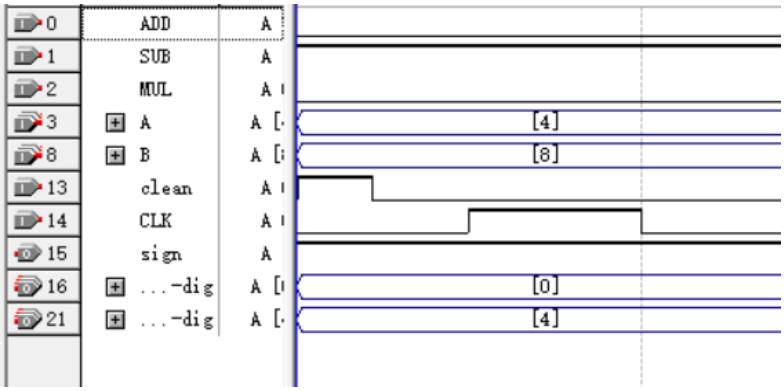
可以看到，十位（蓝色）与个位（红色）输出为 0、8，正确输出了 BCD 码结果。

BCD 计算器(calculator)：

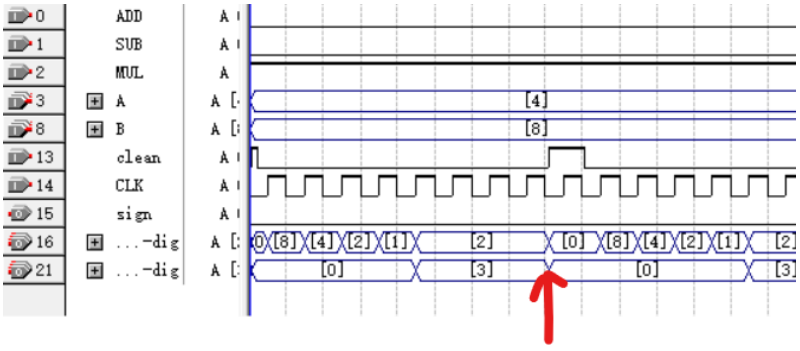
以 A=4、B=8 为例。即计算式分别为 4 + 8、4 – 8、4 x 8。从上至下分别为加法、减法、乘法。



（加法：4+8）



（减法：4-8）



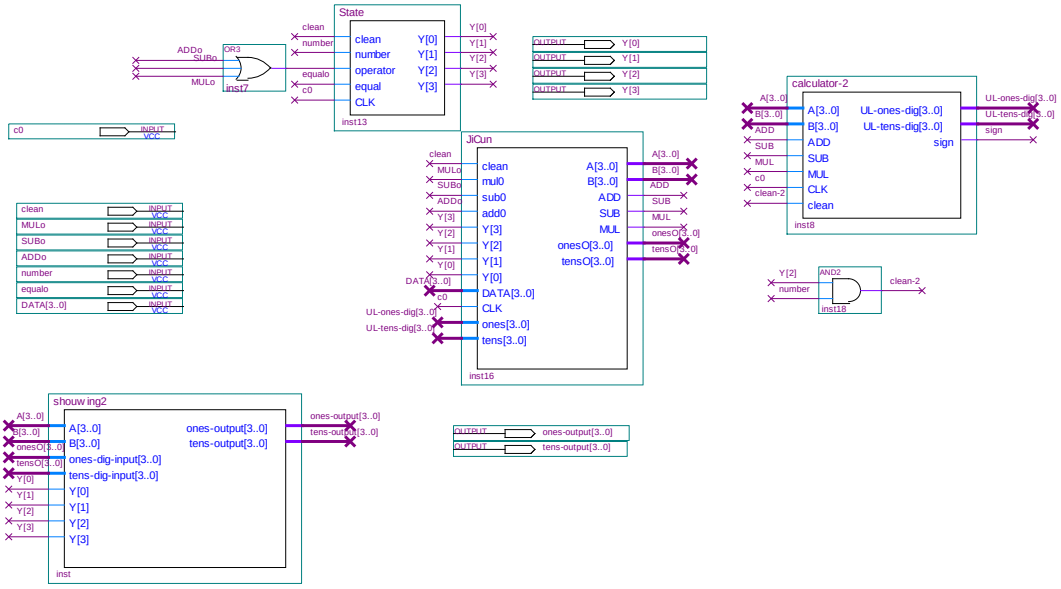
（乘法：4x8）

可以看到，对于输入 A=4、B=8，各功能均正确输出了 BCD 码表示的计算结果。其中，减法功能的 sign（符号）也正确输出了“1”，即负号。

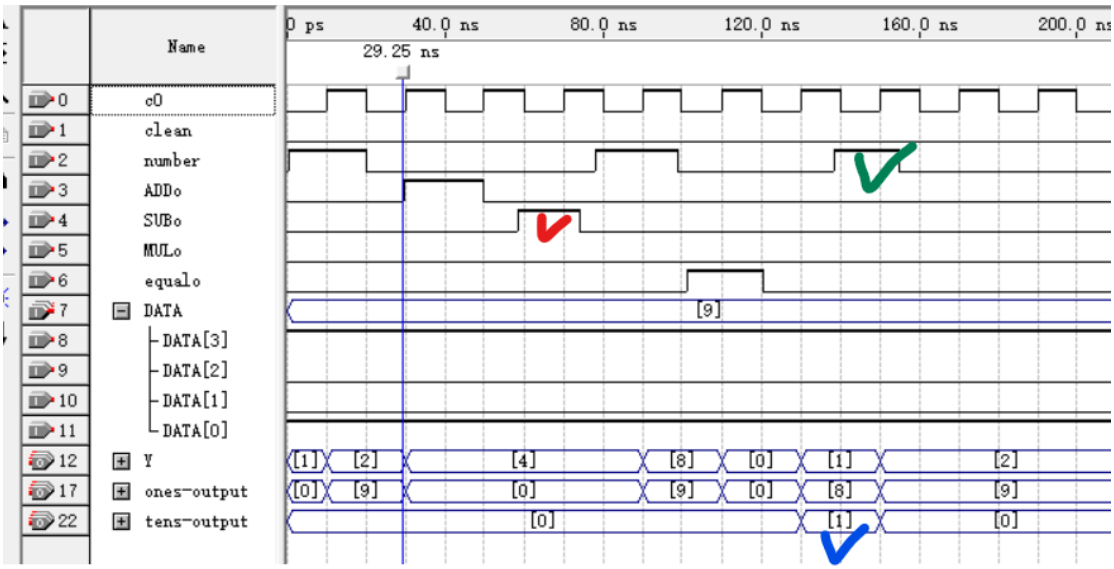
同时，对于乘法功能，若在时钟的低电平时期加入一个 clean 的高电平信号，观察到结果立刻被清零。这表明 clean 的异步清零功能有效。

顶端（cal-UL）：

为便于总和验证，对文件做更改如下。



删去了用于人机交互的部分。



这里以计算式 9 + 9 为例。可以看到，在依次输入 number 信号后，正确实现了状态转换。如果错误输入了信号（红色勾部分），则状态不会跳转。最后，正确输出了计算结果 18（蓝色勾部分）。计算完成后，若再次给进 number 信号，会重新开始计算（绿色勾部分）。

各分部计算式已在上文验证，因此在此仅验证总和功能无误即可。

6. 管脚分配

1		c0	Output					2.5 V (default)	
2		clk_10kHz	Input	PIN_E1	1	B1_N0		2.5 V (default)	Off
3		col[3]	Input	PIN_B14	7	B7_N0		2.5 V (default)	On
4		col[2]	Input	PIN_A14	7	B7_N0		2.5 V (default)	On
5		col[1]	Input	PIN_B13	7	B7_N0		2.5 V (default)	On
6		col[0]	Input	PIN_A13	7	B7_N0		2.5 V (default)	On
7		dout[6]	Output	PIN_D9	7	B7_N0		2.5 V (default)	On
8		dout[5]	Output	PIN_E10	7	B7_N0		2.5 V (default)	On
9		dout[4]	Output	PIN_E8	8	B8_N0		2.5 V (default)	On
10		dout[3]	Output	PIN_D11	7	B7_N0		2.5 V (default)	On
11		dout[2]	Output	PIN_C8	8	B8_N0		2.5 V (default)	On
12		dout[1]	Output	PIN_D8	8	B8_N0		2.5 V (default)	On
13		dout[0]	Output	PIN_E9	7	B7_N0		2.5 V (default)	On
14		key_0_5	Output					2.5 V (default)	
15		key_0_9	Output					2.5 V (default)	
16		key_A	Output					2.5 V (default)	
17		key_B	Output					2.5 V (default)	
18		key_C	Output					2.5 V (default)	
19		key_D	Output					2.5 V (default)	
20		key_data[3]	Output					2.5 V (default)	

24		key_E	Output					2.5 V (default)	
25		key_F	Output					2.5 V (default)	
26		key_pressed	Output					2.5 V (default)	
27		pout[3]	Output	PIN_C14	7	B7_N0		2.5 V (default)	On
28		pout[2]	Output	PIN_D14	7	B7_N0		2.5 V (default)	On
29		pout[1]	Output	PIN_G11	6	B6_N0		2.5 V (default)	On
30		pout[0]	Output	PIN_F11	7	B7_N0		2.5 V (default)	On
31		row[3]	Output	PIN_C16	6	B6_N0		2.5 V (default)	On
32		row[2]	Output	PIN_C15	6	B6_N0		2.5 V (default)	On
33		row[1]	Output	PIN_B16	6	B6_N0		2.5 V (default)	On
34		row[0]	Output	PIN_A15	7	B7_N0		2.5 V (default)	On
35		Y[3]	Output	PIN_B4	8	B8_N0		2.5 V (default)	On
36		Y[2]	Output	PIN_A4	8	B8_N0		2.5 V (default)	On
37		Y[1]	Output	PIN_B5	8	B8_N0		2.5 V (default)	On
38		Y[0]	Output	PIN_A5	8	B8_N0		2.5 V (default)	On

三、实验记录（记录实验具体步骤、原始数据、实验过程、实验中遇到的故障现象、排除故障的过程和方法等）

实验步骤：将程序下载到 FPGA 板，并连接键盘。

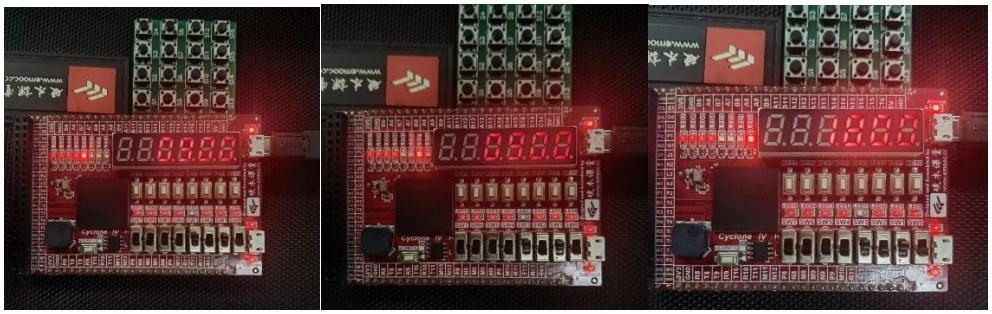
1.**计算功能：**分别输入加法、减法、乘法计算式，每种功能分别验证是否需要进位的两个计算式。观察结果是否正确。

2.**错误输入：**在应当输入数字时输入符号或等号，观察是否会扰乱输入顺序。

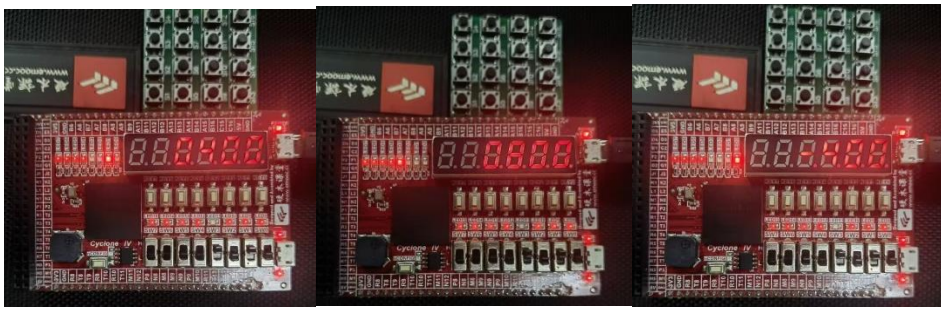
3.**清零功能：**设计时加入了手动清零功能，对应 F 按键。在有数据输入时按下 F 按键，观察是否能实现清零功能。

实验发现，上述功能均能实现。在此各记录一例。

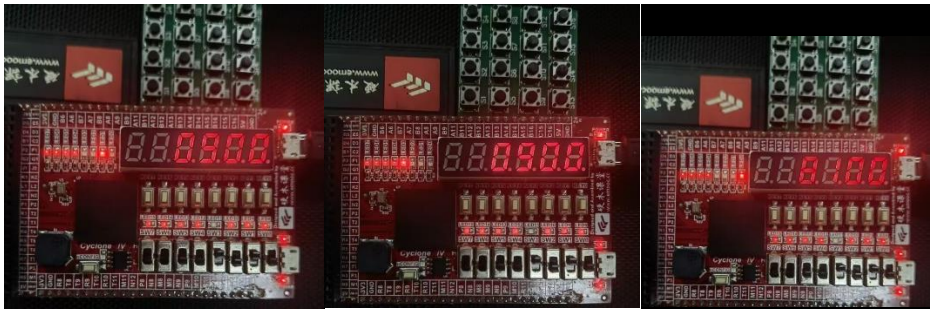
加法：9 + 9 = 18



减法：4 - 8= -4



乘法：9 x 9 = 81



遇到的问题及解决方法：

1.最开始连接到键盘上时，输入现象为：有时按键能改变数码管显示，有时不能；数码管显示的内容不定，有时是 C，有时是 9。

在这种情况下，根据以往经验，首先考虑到了接出状态机。将状态机利用 LED0~3 显示，对应状态即为对应 LED 角标。检测结果发现，状态机在 1~3 间顺序循环，没有达到状态 0。因此，有两种推测：跳过了状态 0，或总控电路的计数有问题。由于总控电路 State 仿真结果正确，再次检查无误，判断为跳过了状态 0。

2.判断为“跳过状态 0”后，由于 State 没有问题，检查芯片的外电路接线逻辑。推演发现，外电路中寄存芯片 clean（所有芯片的清零输入）输入实现的逻辑是：当输入为 number 且为状态 Y[1]时即清零并寄存。由于寄存后会一直保持数据，且此时设计的 State 是通过寄存器控制的，因此直接跳过了状态 0。修改后状态机无误。

四、实验仪器

FPGA 板

五、实验小结（总结实验完成情况，对设计方案和实验结果做必要的讨论，简述实验收获和体会）

本次实验是大作业，也理所当然地是做得最长的一个实验，共计 3 周，比别人多出来的一周是序列检测器割下来的。看了眼自己的报告，嗯，21 页，可能写完这坨就 22 页了，好像第一次报告加上修改的三个实验我也才写了 30 页。虽然有一大坨都是图，但这次显然是也是打字最多的一次。我怎么知道的呢？因为手疼

这次实验可谓倾尽了我学期所学，从自己突发奇想地乱摸索源代码最后真的捣鼓出一个非常好看的负号，到报告的撰写，是本学期各方面的集大成者。为什么提到了写报告呢？因为这次报告我自我感觉良好，嘿嘿。其实是因为这次报告我采用了一些相对来说标准化的方式去写，比如标准化命名（这是在上一个实验里为了打字省事学会的，像写论文一样来一个 ASPRG）、好看的状态图（刚学会用 draw.io 这个网站，真的好好看！）、公式的 1.5 倍行距（因为也在写大物实验论文，自行推导的公式有几十个，密密麻麻很丑，发现改成 1.5 倍行距就很好看了）等等等等，所以真的可以说是本学期各方面，不仅仅是数电实验的集大成

者。

也发现了，其实很多任务都是进行人机交互。这次改负号其实不仅仅是想到这一层就去试了，还有一层原因是，我在自行实验时思考过实验二与本实验的区别。其实不是功能实现难度的提升，主要的难度提升在于人机交互。暑期学校的 C++大作业也如出一辙，程序主体是早就编写好的功能，但为功能加上人机交互的部分，难度甚至比实现基本功能还大。所以我想，既然已经加了那么多人机交互的难度，最后负号还要用 F 表示，好像有点亏。所以我才实践了负号的修改。

六、参考资料

《数字逻辑设计实践 2023 年教学计划 A_V1.0》